

Programming Exercise 11

Vector Distance Calculation

C# Step by Step

1 Introduction to exercise

A *vector* is a mathematical object that represents both magnitude and direction. For example, a projectile fired from a gun barrel can be modeled as a vector because it has both magnitude and direction. An aircraft in flight can be modeled as a vector because it has both magnitude and direction. A parachutist jumping from an airplane can be modeled as a vector because he has both magnitude and direction. The direction of the parachutist is (generally) down, and the magnitude is the velocity of the fall, which (hopefully) will become much slower at some point before he hits the ground.

In science and technology, we frequently have need to compare vectors and compute the closest vectors. For example, a dating service might have a personality profile of 100 points, modeled as a vector, and need to compare the personality profiles of all members to determine the closest matches. A bank may have a customer profile of 1000 points, modeled as a vector, and need to group its customers into similar categories. A biologist may have a genetic profile consisting of 1,000,000 points, modeled as a vector, and need to find the closest match.

How do we calculate the distance between two vectors? There are a number of ways, but the simplest and easiest way is to use what's called *Euclidean distance*, which is nothing more than the practical application of the Pythagorean theorem. Recall that the Pythagorean theorem allows us to compute the length of the hypotenuse of a right triangle, given the length of two sides, x and y , like this:

$$h = \sqrt{x^2 + y^2} \tag{1}$$

Suppose we don't have the length of two sides, but two points. What do we do then? If we consider points on a two dimensional grid, a *point* consists of an x location and a y location, like this: (0,0). For this point, the x location is at 0 on the X axis, and the y location is at 0 on the Y axis. When we have two points, we simply subtract the second point from the first point. If we have two points, (1,1) and (4,5), we do it like this:

$$x = x_2 - x_1 \implies x = 4 - 1 \implies x = 3 \tag{2}$$

$$y = y_2 - y_1 \implies y = 5 - 1 \implies y = 4 \tag{3}$$

In this case, the length of x is 3, the length of y is 4, and the distance between the two points is 5. We calculate this by the Pythagorean formula.

$$\begin{aligned} L &= \sqrt{x^2 + y^2} \\ L &= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \\ L &= \sqrt{(4 - 1)^2 + (5 - 1)^2} \\ L &= \sqrt{3^2 + 4^2} \\ L &= \sqrt{9 + 16} \\ L &= \sqrt{25} \\ L &= 5 \end{aligned}$$

Now, suppose we have three two-dimensional vectors, which we can think of as points, like this:

1. (1, 1)
2. (4, 5)
3. (5, 4)

What is the closest pair of points? See figure 1.

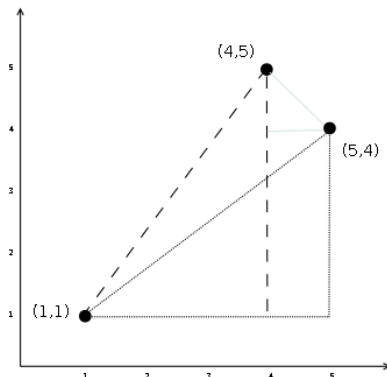


Figure 1: Calculate distance between three points

We can compare each pair of points and compute the Euclidean distance, like this:

$$\begin{aligned}\sqrt{(4-1)^2 + (5-1)^2} &= 5 \\ \sqrt{(5-1)^2 + (4-1)^2} &= 5 \\ \sqrt{(4-5)^2 + (5-4)^2} &= \sqrt{2} \approx 1.4142\end{aligned}$$

The points (5,4) and (4,5) are much closer to each other than (1,1) is to (5,4) or (1,1) is to (4,5). This is obvious by looking at figure 1.

What happens if we have vectors of more than two points, i.e., we have points in three-dimensional space, or higher? For example, our points might look like (423, 632, 293). We simply expand the Pythagorean formula. This is the formula for three-dimensional space:

$$h = \sqrt{x^2 + y^2 + z^2} \quad (4)$$

For n -dimensional space, we can expand the formula to include all the dimensions that we have, like this:

$$h = \sqrt{(p_1)^2 + (p_2)^2 + \dots + (p_n)^2} \quad (5)$$

2 First part, 70 points

In the first two parts, we will find the two closest two-dimensional vectors, modeled as points in a two-dimensional space. Write a console application that creates some sort of data structure that contains two integers, x and y , as random numbers between 1 and 100. Do not use the built-in class `Point`, rather, build your own data structure. Please note, we will think of these objects as *points*, but in reality they are *vectors*. Then, create some sort of collection that contains 100 of these points.

3 Second part, 80 points

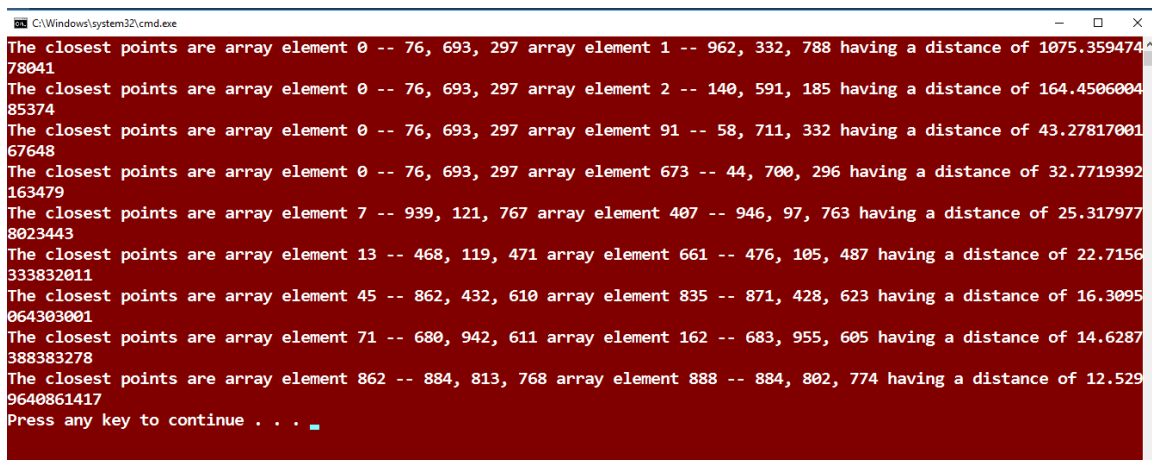
Write a function that takes two points and computes the Euclidean distance between them. Then, write a function that iterates through your collection, compares each point to every other point and reports the closest two points.

4 Third part, 90 points

In the second two parts, we will find the two closest three-dimensional vectors, modeled as points in a three-dimensional space. Write a console application that creates some sort of data structure that contains three integers, x , y , and z , as random numbers between 1 and 1000. Do not use any built-in class but build your own data structure. Please note, we will think of these objects as *points*, but in reality they are *vectors*. Then, create some sort of collection that contains 1000 of these points.

5 Fourth part, 100 points

Write a function that takes two three-dimensional points and computes the Euclidean distance between them. Then, write a function that iterates through your collection, compares each point to every other point and reports the closest two points. Your output for the three point vectors might look like figure 2.



```

C:\Windows\system32\cmd.exe
The closest points are array element 0 -- 76, 693, 297 array element 1 -- 962, 332, 788 having a distance of 1075.359474
78041
The closest points are array element 0 -- 76, 693, 297 array element 2 -- 140, 591, 185 having a distance of 164.4506004
85374
The closest points are array element 0 -- 76, 693, 297 array element 91 -- 58, 711, 332 having a distance of 43.27817001
67648
The closest points are array element 0 -- 76, 693, 297 array element 673 -- 44, 700, 296 having a distance of 32.7719392
163479
The closest points are array element 7 -- 939, 121, 767 array element 407 -- 946, 97, 763 having a distance of 25.317977
8023443
The closest points are array element 13 -- 468, 119, 471 array element 661 -- 476, 105, 487 having a distance of 22.7156
333832011
The closest points are array element 45 -- 862, 432, 610 array element 835 -- 871, 428, 623 having a distance of 16.3095
064303001
The closest points are array element 71 -- 680, 942, 611 array element 162 -- 683, 955, 605 having a distance of 14.6287
388383278
The closest points are array element 862 -- 884, 813, 768 array element 888 -- 884, 802, 774 having a distance of 12.529
9640861417
Press any key to continue . . .

```

Figure 2: Result of vector distance calculation for three points

6 Code template

You do *not* have to use the following code template. However, it may prove useful as a starting point. This is the `Main()` method of my implementation if this exercise. I created two structs named `Datum2` and `Datum3`. You can create them as classes if you like. For each, I wrote a void method named `CalcDist`, which takes no parameters. You must write the code creating the structs (or classes) and the appropriate method(s).

```

1 static void Main(string[] args)
2 {
3     Console.WriteLine("This is the Vector Distance exercise");
4     bool again = true;
5     while (again)
6     {
7         Console.Write("\nEnter 2 to calculate 2 element vector , " +
8             "\n3 to calculate 3 element vector , 9 to quit : ");
9         string input = Console.ReadLine();

```

```

10         if (input.Equals("2"))
11         {
12             Console.WriteLine("\ntwo_element_vector");
13             Datum2 d2 = new Datum2();
14             d2.CalcDist();
15         }
16         else if (input.Equals("3"))
17         {
18             Console.WriteLine("\nthree_element_vector");
19             Datum3 d3 = new Datum3();
20             d3.CalcDist();
21         }
22         else if (input.Equals("9"))
23             again = false; //to exit
24         else
25             Console.WriteLine("\nDidn't recognize input");
26     }
27     System.Environment.Exit(0);
28 }

```

The document that I would turn in as an answer to this exercise would look like this, with ... representing the code that you would write.

```

1  using System;
2
3  namespace VectorDistance
4  {
5      struct Datum2
6      {
7          //your code here
8          ...
9      }
10
11     struct Datum3
12     {
13         //your code here
14         ...
15     }
16
17     class Program
18     {
19         static void Main(string[] args)
20         {
21             //your code here
22             ...
23         }
24     }
25 }

```